



AF & IFW

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

In re application of: Plummer et al.

Attorney Docket No.: SUN1P802/P5257

Application No.: 09/841,759

Examiner: Tang, Kuo Liang J.

Filed: April 24, 2001

Group: 2191

Title: METHOD AND APPARATUS FOR  
REWRITING BYTECODES TO MINIMIZE  
RUNTIME CHECKS

**CERTIFICATE OF MAILING**

I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as first-class mail on June 1, 2005 in an envelope addressed to the Commissioner for Patents, Mail Stop Appeal Brief-Patents, P.O. Box 1450 Alexandria, VA 22313-1450.

Signed: \_\_\_\_\_

Dilora Haddad

**APPEAL BRIEF TRANSMITTAL  
(37 CFR 192)**

Mail Stop Appeal Brief-Patents  
Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

Sir:

This brief is in furtherance of the Notice of Appeal filed in this case on April 1, 2005.

This application is on behalf of

☐

Small Entity

☒

Large Entity

Pursuant to 37 CFR 1.17(f), the fee for filing the Appeal Brief is:

☐

\$170.00 (Small Entity)

☒

\$340.00 (Large Entity)

☐

Applicant(s) hereby petition for a \_\_\_\_\_ extension(s) of time to under 37 CFR 1.136.

If an additional extension of time is required, please consider this a petition therefor.

☐

An extension for \_\_\_\_\_ months has already been secured and the fee paid therefor of \$ \_\_\_\_\_ is deducted from the total fee due for the total months of extension now requested.

☒ Applicant(s) believe that no (additional) Extension of Time is required; however, if it is determined that such an extension is required, Applicant(s) hereby petition that such an extension be granted and authorize the Commissioner to charge the required fees for an Extension of Time under 37 CFR 1.136 to Deposit Account No. 500388 (Order No. SUN1P802).

Total Fee Due:

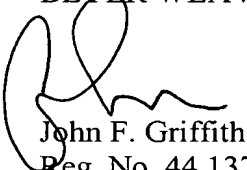
Appeal Brief fee \$340.00

Total Fee Due \$340.00

☒ Enclosed is Check No.10687 in the amount of \$340.00.

☒ Charge any additional fees or credit any overpayment to Deposit Account No. 500388, (Order No. SUN1P802). Two copies of this transmittal are enclosed.

Respectfully submitted,  
BEYER WEAVER & THOMAS, LLP

  
John F. Griffith  
Reg. No. 44,137

P.O. Box 70250  
Oakland, CA 94612-0250  
(510) 663-1100



**PATENT**

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE  
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES**

---

**EX PARTE PLUMMER et al.**

---

**Application for Patent**

**Filed 04/24/01**

**Serial No. 09/841,759**

**FOR:**

**METHOD AND APPARATUS FOR REWRITING BYTECODES TO MINIMIZE  
RUNTIME CHECKS**

---

**APPEAL BRIEF**

---

**CERTIFICATE OF MAILING**

I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as first-class mail on June 1, 2005 in an envelope addressed to Mail Stop Appeal Brief - Patents, Commissioner for Patents, P.O. Box 1450 Alexandria, VA 22313-1450.

Signed: 

Dilora Haddad

06/06/2005 MAHMED1 00000023 500388 09841759  
01 FC:1402 160.00 DA 340.00 OP

**BEYER WEAVER & THOMAS, LLP  
Attorneys for Appellants**

## TABLE OF CONTENTS

	<u>Page No.</u>
<b>I. REAL PARTY IN INTEREST</b>	<b>1</b>
<b>II. RELATED APPEALS AND INTERFERENCES</b>	<b>1</b>
<b>III. STATUS OF CLAIMS</b>	<b>1</b>
<b>IV. STATUS OF AMENDMENTS</b>	<b>1</b>
<b>V. SUMMARY OF CLAIMED SUBJECT MATTER</b>	<b>1</b>
<b>VI. GROUND OF REJECTION TO BE REVIEWED ON APPEAL</b>	<b>3</b>
<b>VII. ARGUMENT</b>	<b>3</b>
<b>A. The Long and Seshadri References Fail to Support a Rejection of Claims 1, 7 and 12 Under 35 U.S.C. § 103(a).</b>	<b>3</b>
<b>1. The Long Reference Fails to Disclose or Reasonably Suggest a Preloader Arranged to Rewrite a Bytecode to a New Bytecode Which Indicates That at Least One of a Class and a Superclass Requires Execution of a Static_INITIALIZER.</b>	<b>4</b>
<b>2. The Seshadri Reference Fails to Disclose or Reasonably Suggest a Preloader Arranged to Rewrite a Bytecode to a New Bytecode Which Indicates That at Least One of a Class and a Superclass Requires Execution of a Static_INITIALIZER.</b>	<b>7</b>
<b>B. The Long and Seshadri References Fail to Support a Rejection of Claims 3, 9 and 14 Under 35 U.S.C. § 103(a).</b>	<b>10</b>
<b>C. The Long and Seshadri References Fail to Support a Rejection of Claims 4, 5, 10, 11, 15 and 16 Under 35 U.S.C. § 103(a).</b>	<b>12</b>
<b>D. The Long and Seshadri References Fail to Support a Rejection of Claim 6 Under 35 U.S.C. § 103(a).</b>	<b>14</b>
<b>E. Conclusion</b>	<b>16</b>
<b>VIII. CLAIMS APPENDIX</b>	<b>18</b>

## **I. REAL PARTY IN INTEREST**

The real party in interest is Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303.

## **II. RELATED APPEALS AND INTERFERENCES**

There are no related appeals, interferences, or judicial proceedings known to the Appellants.

## **III. STATUS OF CLAIMS**

There are a total of 13 claims pending in this application (claims 1, 3-7, 9-12 and 14-16), all of which were submitted with the application as filed. All of the pending claims were rejected under 35 U.S.C. § 103(a) as obvious in view of Long, U.S. Patent No. 6,691,307 and Seshadri, U.S. Patent No. 6,658,421 ("Seshadri"). Appellants appeal all of the rejections of the pending claims and respectfully request reversal of the rejections.

## **IV. STATUS OF AMENDMENTS**

Amendments to claims 1, 7 and 12 were submitted in the response of December 28, 2004, following the final Office Action of December 2, 2004. These amendments were entered as indicated in the Advisory Action mailed February 22, 2005.

## **V. SUMMARY OF CLAIMED SUBJECT MATTER**

Aspects of the present invention relate to methods, apparatus, and computer program products for reducing the number of runtime checks performed during the execution of a virtual machine. (Application as filed, page 6, lines 3-4; page 8, lines 10-18). According to

one aspect of the present invention, as shown in Fig. 1, a computer system 110 includes a preloader 118 or bytecode rewriter, a compiler 122, and a virtual machine 126. (Application as filed, page 2, line 27 through page 3, line 7). In Fig. 1, the compiler 122 is arranged to accept a source file, such as “ramjava.c,” generated by the preloader 118 as input and to produce an object file, such as “romjava.o,” and the virtual machine 126 is arranged to execute the object file. (*Id.*).

As shown in Figs. 1 and 2, the preloader 118 is arranged to determine whether a bytecode makes an active reference to a class which requires an execution of a static initializer such as a “<clinit>” method, in steps 204 and 208, and is also arranged to determine if the class has a superclass which requires the execution of the static initializer, in step 212. (Application as filed, page 8, line 23 through page 9, line 2). In one embodiment, as shown in Fig. 2, the preloader 118 rewrites the bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of the static initializer, in step 216, when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer, following step 208, or the class which has the superclass which requires the execution of the static initializer, following step 212. (Application as filed, page 9, lines 9-18). In one embodiment, in step 216, the new bytecode explicitly indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer, or the class which has the superclass which requires the execution of the static initializer. (Application as filed, page 9, lines 20-26).

According to another aspect of the present invention, the bytecode rewriter is arranged to determine whether a bytecode is associated with a scalar field or an object reference field. (Application as filed, page 10, lines 18-20). The bytecode rewriter is arranged to rewrite the

bytecode to identify the bytecode as being associated with the scalar field when the bytecode is associated with the scalar field. (*Id.*; page 10, lines 22-26; page 11, lines 4-6). The bytecode rewriter is further arranged to rewrite the bytecode to identify the bytecode as being associated with the object reference field when the bytecode is associated with the object reference field. (Application as filed, page 10, lines 22-30).

## **VI. GROUND OF REJECTION TO BE REVIEWED ON APPEAL**

The rejection presented for review is as follows:

Claims 1-16 were rejected under 35 U.S.C. § 103(a) as obvious in view of Long and Seshadri. Claims 2, 8 and 13 were cancelled in the amendment after final. Therefore, the outstanding rejection of claims 1, 3-7, 9-12 and 14-16 is appealed.

## **VII. ARGUMENT**

With respect to the rejection above, the rejected claims do not stand or fall together. Claims 1, 7 and 12 are presented as a first group, claims 3, 9 and 14 are presented as a second group, and claims 4, 5, 10, 11, 15 and 16 are presented as a third group. Claim 6 is presented separately. The second group, the third group of claims, and claim 6 are each separately patentable for the reasons below.

### **A. The Long and Seshadri References Fail to Support a Rejection of Claims 1, 7 and 12 Under 35 U.S.C. § 103(a).**

Long and Seshadri were cited in the Final Office Action of December 2, 2004 to support the rejection of claims 1, 7 and 12 as obvious under 35 U.S.C. § 103(a). This rejection should be reversed for the reasons below.

**1. The Long Reference Fails to Disclose or Reasonably Suggest a Preloader Arranged to Rewrite a Bytecode to a New Bytecode Which Indicates That at Least One of a Class and a Superclass Requires Execution of a Static\_INITIALIZER.**

Claim 1, by way of example, defines a computer system comprising a preloader. The preloader produces a source file. A compiler is arranged to accept the source file and produce an object file. A virtual machine is arranged to execute the object file. The preloader of the computer system defined in claim 1 is arranged to:

determine whether a bytecode makes an active reference to a class which requires an execution of a static initializer,

determine if the class has a superclass which requires the execution of the static initializer, wherein the preloader produces a source file,

rewrite the bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer . . .

(Emphasis Added).

The preloader of claim 1 provides the features of making the determinations recited above, and “rewrit[ing] the bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer.” In this way, in one embodiment, the preloader can reduce the number of runtime checks for static initializers during execution of the object file at the virtual machine.

It is acknowledged that the primary reference (Long) discloses a computing system that includes a preloader, a compiler and a virtual machine. However, Long does not disclose any of the preloader functionality required by claim 1. Specifically, Long does not disclose a

preloader that is arranged to:

determine whether a bytecode makes an active reference to a class which requires an execution of a static initializer,

determine if the class has a superclass which requires the execution of the static initializer, wherein the preloader produces a source file, or

rewrite the bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer . . .

The outstanding rejections acknowledge that Long does not disclose either of the determining functionalities and instead attempts to rely on the secondary Seshadri reference in an effort to meet the claimed features. As will be discussed in more detail below, it is respectfully submitted that Seshadri does not suggest these features in the context of the claimed invention.

The outstanding rejection then asserts that Long teaches the bytecode rewriting feature. However, Long does not even remotely suggest the claimed preloader functionality of:

rewrit[ing] the bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer . . .

The Final Office Action asserted that the preloader bytecode rewriting functionality is disclosed in the form of Long's preloader 172 and runtime system 174 (col. 7:22-33, Fig. 9A). (See, page 5, lines 1-6, which discusses the language of previously presented claim 2 which has now been added to claim 1). The teachings of Long simply do not support this assertion.

Specifically, the referenced section of Long (Col. 7, lines 22-33) reads:

Turning now to FIG. 9A, a block diagram that illustrates pre-loading Java classfiles in accordance with one embodiment of the present invention is presented. The preloader 172 loads a classfile 170 and organizes code within the classfile according to native endianness. The preloader 172 outputs code and data structures for the runtime system 174. The output maybe in the form of .c files or linkable object files. The .c files are compiled into object files and the object files are linked together into the runtime system 174 executable image either at build time or at runtime via dynamic linking. The runtime system 174 makes calls to the native operating system 176.

Clearly, the above passage does not even remotely suggest the claimed feature of a preloader arranged to:

rewrite the bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer . . .

Indeed the cited passage does not even discuss static initializers, and says nothing of the functionality of rewriting a bytecode to indicate that at least one of a class and a superclass requires execution of a static initializer. Since the outstanding rejection does not point to any teaching in either the primary or secondary reference that discloses or remotely suggests such a feature, it is respectfully submitted that the outstanding rejection is improper and should be withdrawn for at least this reason.

For completeness, it is noted that the Long patent is only concerned with interpreter optimization for native endianness (col. 4, lines 9-10), not eliminating unnecessary runtime checks for static initializers. To address the problem of interpreter optimization, Long suggests one technique for reversing the endian order of a classfile (e.g., big endian to little endian, or vice-versa) for interpretation. (col. 5, line 66 through col. 6, line 3). In particular, “[t]he preloader 172 loads a classfile 170 and organizes code within the classfile according to native endianness . . . and outputs code and data structures for the runtime system 174.” (col. 7,

lines 24-28). In this way, a runtime system can load and execute code without having to identify and resolve inconsistencies in the endian order of bytecodes.

The teachings in the Long patent all relate to this reordering of the endian format of a bytecode. Long is simply not concerned with the problem of redundant runtime checks for static initializers and does not in any way suggest rewriting a bytecode to indicate that at least one of a class and a superclass requires execution of a static initializer.

**2. The Seshadri Reference Fails to Disclose or Reasonably Suggest a Preloader Arranged to Rewrite a Bytecode to a New Bytecode Which Indicates That at Least One of a Class and a Superclass Requires Execution of a Static Initializer.**

As discussed above, the outstanding rejection acknowledges that Long does not disclose either of the determining functionalities and instead attempts to rely on the secondary Seshadri reference in an effort to meet the claimed features. The Final Office Action stated, on pages 2-3, that “Seshadri in an analogous art teaches ‘a preloader arranged to, determine whether a bytecode (E.g. see col. 4:56, invokestatic) makes an active reference to a class which requires an execution of a static initializer, determine if the class has a superclass (E.g. see col. 4:66 and col. 5:5) which requires the execution of the static initializer, wherein the preloader produces a source file.’ (E.g. see TABLE 2 at col. 12 and see col. 4:54 to col. 5:5).” Appellants disagree with this assessment of Seshadri.

Seshadri fails to disclose or suggest the features of claim 1, including a preloader arranged to:

determine whether a bytecode makes an active reference to a class which requires an execution of a static initializer,

determine if the class has a superclass which requires the execution of the static initializer, wherein the preloader produces a source file, . . .

Seshadri describes encoding characterizing indicia, e.g., a method block table signature, into class metadata during compilation, (col. 4, lines 2-5, 22-26), that is, when compiling an invokestatic bytecode. (col. 4, lines 55-57). Also, Seshadri mentions encoding an instance data signature for an implicit referent class which is a superclass of said referring class. (col. 4, lines 64-66). Nowhere, however, does Long disclose or suggest determining whether the invokestatic bytecode makes an active reference to a class which requires an execution of a static initializer, or determining whether any implicit referent class requires execution of the static initializer.

In addition, Seshadri does not disclose or suggest a preloader arranged to perform the above-quoted functionality of claim 1. Instead, Seshadri's teachings relate to the output of a compiler, that is, detecting binary compatibility in compiled object code. (col. 4, line 55). According to Seshadri, the characterizing indicia are encoded into class metadata during compilation. (col. 4, lines 2-5, 22-26). Seshadri explains that the method block table signature is encoded when compiling an "invokestatic bytecode." (col. 4, lines 55-56). Following the teachings of Seshadri, the source file would have to be generated and output to the compiler for compiling, before any processing would begin. Then, the encoding would be performed during compilation of the source file to produce an object file. Therefore, the encoding

techniques described by Seshadri could not be performed by “a preloader ... wherein the preloader produces a source file,” as recited in claim 1.

The Final Office Action cites Table 2 and col. 12, lines 25-36 as showing a preloader but, in fact, the term preloader is not mentioned at all in the quoted passages. Moreover, there is no disclosure or suggestion of the preloader defined in claim 1 as arranged to:

determine whether a bytecode makes an active reference to a class which requires an execution of a static initializer,

determine if the class has a superclass which requires the execution of the static initializer, wherein the preloader produces a source file, . . .

In addition, nowhere does Seshadri suggest a preloader arranged to rewrite a bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of a static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer, as recited in claim 1. The generating and encoding of signatures described by Seshadri is for detecting binary incompatibility in object code, (col. 4, lines 9-21), not eliminating unnecessary runtime checks for static initializers. Seshadri is not concerned with rewriting a bytecode to a new bytecode which indicates that at least one of a class and a superclass requires execution of a static initializer, and fails to offer any teaching in this regard.

Because Seshadri fails to disclose or suggest the above-quoted features of the computer system of claim 1, namely: (1) a preloader arranged to, (2) determine whether a bytecode makes an active reference to a class which requires an execution of a static initializer, (3) determine if the class has a superclass which requires the execution of the static initializer, (4) wherein the preloader produces a source file, and (5) rewrite the bytecode to a new bytecode which indicates that at least one of the class and the superclass requires

execution of the static initializer, Seshadri fails to support an obviousness rejection of claim 1 under 35 U.S.C. § 103(a), considered alone or in combination with Long.

Lastly, Appellants note that there is no suggestion or motivation to combine the teachings of Long with Seshadri, because the respective problems addressed by Long and Seshadri are different. Long's teachings pertain to the endianness of code, that is, reversing the order of code to conform to big endian or little endian format. Seshadri, on the other hand, has to do with detecting binary incompatibility in compiled object code, (col. 4, lines 9-21). There is no overlap in the subject matter of these problems to be solved, nor do the respective problems of Long and Seshadri disclose the features of the preloader defined in claim 1. Based on the cited references, there would have been no teaching or motivation to those of ordinary skill in the art to combine Long with Seshadri to address a different problem of eliminating redundant runtime checks for static initializers. Therefore, the combination of Long and Seshadri is improper under a 35 U.S.C. §103(a) analysis, and should be reversed.

Independent claims 7 and 12 incorporate similar features as claim 1. Thus, the cited references, taken alone or in combination, fail to support a rejection of these claims for the same reasons as claim 1. The rejection of these claims should be reversed.

**B. The Long and Seshadri References Fail to Support a Rejection of Claims 3, 9 and 14 Under 35 U.S.C. § 103(a).**

Long and Seshadri were cited in the Final Office Action to support the rejection of claims 3, 9 and 14 as obvious under 35 U.S.C. § 103(a). This rejection should be reversed for the reasons below.

Claims 3, 9 and 14 are dependent upon claims reciting the features described in section VII.A. above. Thus, Long and Seshadri fail to support rejections of these dependent claims

for at least the same reasons as the independent claims on which they are based. In addition, claims 3, 9 and 14 recite at least one additional feature which Long and Seshadri fail to disclose or suggest. Claim 3, by way of example, further defines the preloader of claim 1 as arranged to:

rewrite the bytecode to a new bytecode which explicitly indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer.

(Emphasis added.)

The preloader defined in claim 3 not only provides the feature of rewriting the bytecode in the manner recited in claim 1, but also provides that the new bytecode explicitly indicate that at least one of the class and the superclass requires execution of the static initializer. Thus, in one embodiment, opcodes which need to be checked to determine if it is necessary to run a static initializer can be explicitly identified, for example, with a scalar value, to significantly reduce the number of runtime checks. (Application as filed, page 9, lines 20-26).

Long fails to make any mention of rewriting a bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of the static initializer, as explained in section VII.A. above. Again, Long's teachings all pertain to the reordering of the endian format of a bytecode. Long is simply not concerned with the problem of redundant runtime checks for static initializers. Thus, with regard to claim 3, it is clear that Long fails to teach a new bytecode which explicitly makes such an indication, for the same reasons that Long fails to disclose or suggest the rewriting of a bytecode to a new bytecode as defined in claim 1.

Seshadri, considered alone or in combination with Long, fails to support an obviousness rejection of claim 3. The generating and encoding of signatures described by

Seshadri is for detecting binary incompatibility in object code, (col. 4, lines 9-21), not eliminating unnecessary runtime checks for static initializers. Seshadri does not describe rewriting a bytecode to a new bytecode which indicates that at least one of a class and a superclass requires execution of a static initializer, much less explicitly indicating that at least one of the class and the superclass requires execution of the static initializer, and fails to offer any teaching in this regard.

Accordingly, Long and Seshadri, considered alone or in combination, fail to support an obviousness rejection of claim 3 under 35 U.S.C. § 103(a). This rejection should be reversed.

Claims 9 and 14 incorporate similar features as claim 3. Thus, the cited references, taken alone or in combination, fail to support a rejection of these claims for at least the same reasons as claim 3. The rejection of these claims should be reversed.

**C. The Long and Seshadri References Fail to Support a Rejection of Claims 4, 5, 10, 11, 15 and 16 Under 35 U.S.C. § 103(a).**

Long and Seshadri were cited in the Final Office Action to support the rejection of claims 4, 5, 10, 11, 15 and 16 as obvious under 35 U.S.C. § 103(a). This rejection should be reversed for the reasons below.

Claims 4, 5, 10, 11, 15 and 16 are dependent upon claims reciting the features described in section VII.A. above. Thus, Long and Seshadri fail to support rejections of these dependent claims for at least the same reasons. In addition, claims 4, 5, 10, 11, 15 and 16 recite at least one additional feature which Long and Seshadri fail to disclose or suggest.

Claim 4, by way of example, further defines the preloader of claim 1 as arranged to:

rewrite the bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which has the superclass which requires the execution of the static initializer.

(Emphasis added.)

In one embodiment, the preloader defined in claim 4 again reduces the number of runtime checks for static initializers, not only for bytecodes making an active reference to a class requiring execution of a static initializer, but also for bytecodes which make an active reference to the class which has the superclass which requires the execution of the static initializer.”

Long suggests a technique for reversing the endian order of a classfile (e.g., big endian to little endian) for interpretation, but fails to disclose or suggest making any determination as to whether a bytecode makes an active reference to a class which requires execution of a static initializer, much less whether the class has a superclass which requires the execution of the static initializer. (Final Office Action, page 4, lines 7-10). Without making such a determination, Long provides no disclosure or suggestion to those of ordinary skill in the art to “rewrite the bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which has the superclass which requires the execution of the static initializer.” Long’s teachings clearly fall short in this regard.

As mentioned above, Seshadri describes generating and encoding signatures for detecting binary incompatibility in object code, (col. 4, lines 9-21), not eliminating unnecessary runtime checks for static initializers. Seshadri provides no teaching with respect to rewriting a bytecode to a new bytecode which indicates that at least one of a class and a superclass requires execution of a static initializer, much less doing so “when it is determined

that the bytecode makes the active reference to the class which has the superclass which requires the execution of the static initializer.”

Therefore, Seshadri, considered alone or in combination with Long, fails to support an obviousness rejection of claim 4.

Claims 5, 10, 11, 15 and 16 incorporate similar features as claim 4. Thus, the cited references, taken alone or in combination, fail to support a rejection of these claims for at least the same reasons as claim 4. The rejection of these claims should be reversed.

**D. The Long and Seshadri References Fail to Support a Rejection of Claim 6 Under 35 U.S.C. § 103(a).**

Long and Seshadri were cited in the Final Office Action to support the rejection of claim 6 as obvious under 35 U.S.C. § 103(a). This rejection should be reversed for the reasons below.

Claim 6 defines a computer system comprising a bytecode rewriter. The bytecode rewriter is associated with producing a source file. A compiler is arranged to accept the source file and produce an object file. A virtual machine is arranged to execute the object file. The bytecode rewriter of the computer system defined in claim 6 is arranged to:

determine whether a bytecode is associated with a scalar field or an object reference field,

rewrite the bytecode to identify the bytecode as being associated with the scalar field when the bytecode is associated with the scalar field,

rewrite the bytecode to identify the bytecode as being associated with the object reference field when the bytecode is associated with the object reference field . . .

(Emphasis Added).

The Final Office Action stated, on page 6, paragraph 3, that “Long does not explicitly disclose a bytecode rewriter arranged to, determine whether a bytecode is associated with a scalar field or an object reference field, rewrite the bytecode to identify the bytecode as being associated with the scalar field when the bytecode is associated with the scalar field, rewrite the bytecode to identify the bytecode as being associated with the object reference field when the bytecode is associated with the object reference field . . .” Appellants agree with this assessment of Long. As mentioned above, Long’s teachings relate to reversing the endian format of code, not rewriting a bytecode to identify the bytecode as being associated with a scalar field or an object reference field.

The Final Office Action further stated, on page 6, paragraph 3, that Seshadri teaches the above-quoted features of claim 4 at TABLE 2 at col. 12 and col. 4:54 to col. 5:5. Appellants disagree with this assessment of Seshadri.

Seshadri describes techniques for detecting binary compatibility in compiled object code. (col. 3, lines 63-64). According to Seshadri, characterizing indicia, e.g., a method block table signature, is encoded into class metadata during compilation. (col. 4, lines 2-5, 22-26). Seshadri discusses encoding the signature when compiling an invokestatic bytecode. (col. 4, lines 55-57). Nowhere, however, does Seshadri suggest rewriting the bytecode to identify the bytecode as being associated with the scalar field when the bytecode is associated with the scalar field, as recited in claim 6.

Seshadri explains a conventional procedure for loading classes in a Java virtual machine (col. 12, table 2). In col. 13, lines 10-15, Seshadri mentions that a reference to an instance method 127 in a particular class in an invokevirtual or invokespecial bytecode can be replaced by an index to a method table 212. However, Seshadri makes no mention of rewriting the bytecode to identify the bytecode as being associated with a scalar field when the

bytecode is associated with the scalar field. Claim 6, on the other hand, defines a computer system which provides improved computational speed and efficiency by not only “determin[ing] whether a bytecode is associated with a scalar field or an object reference field,” but also “rewrit[ing] the bytecode to identify the bytecode as being associated with the object reference field when the bytecode is associated with the object reference field.”

Because Long and Seshadri both fail to disclose or suggest: (1) determining whether a bytecode is associated with a scalar field or an object reference field, and (2) rewriting the bytecode to identify the bytecode as being associated with the object reference field when the bytecode is associated with the object reference field, the teachings of Seshadri would fail to cure the deficiencies of Long even if Seshadri could somehow be combined with Long. The cited art fails to support an obviousness rejection of claim 6 under 35 U.S.C. § 103(a), considered alone or in combination. Therefore, this rejection should be reversed.

#### **E. Conclusion**

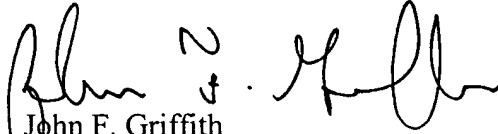
In view of the foregoing, all of the claim rejections under 35 U.S.C. § 103(a) based on the Long and Seshadri references cannot stand for at least the reasons discussed. The Examiner’s reliance on Long and Seshadri is misplaced, because the Examiner has failed to establish a prima facie case of obviousness based on those references.

In view of the foregoing, Appellants respectfully request that the Board reverse the Examiner’s rejection of all pending claims. In addition, Appellants believe all claims now pending in this application are in condition for allowance. The issuance of a formal Notice of Allowance at an early date is respectfully requested.

Enclosed is the required fee for an appeal brief under 37 C.F.R. §1.17(c) in the amount of \$340.

Respectfully Submitted,

BEYER WEAVER & THOMAS, LLP

A handwritten signature in black ink, appearing to read "John F. Griffith", is written over the printed name.

John F. Griffith

Registration No. 44,137

BEYER WEAVER & THOMAS, LLP  
P.O. Box 70250  
Oakland, CA 94612-0250  
(510) 663-1100

## VIII. CLAIMS APPENDIX

### CLAIMS ON APPEAL

1. A computer system comprising:  
a preloader arranged to,  
determine whether a bytecode makes an active reference to a class which requires an execution of a static initializer,  
determine if the class has a superclass which requires the execution of the static initializer, wherein the preloader produces a source file,  
rewrite the bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer;  
a compiler coupled to the preloader arranged to accept the source file as input and produce an object file; and  
a virtual machine coupled to the compiler arranged to execute the object file.
2. (Canceled).
3. A computer system according to claim 1 wherein the preloader is further arranged to:  
rewrite the bytecode to a new bytecode which explicitly indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined

that the bytecode makes the active reference to the class which requires the execution of the static initializer.

4. A computer system according to claim 1 wherein the preloader is further arranged to:

rewrite the bytecode to a new bytecode which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which has the superclass which requires the execution of the static initializer.

5. A computer system according to claim 1 wherein the preloader is further arranged to:

rewrite the bytecode to a new bytecode which explicitly indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which has the superclass which requires the execution of the static initializer.

6. A computer system comprising:

a bytecode rewriter arranged to,

determine whether a bytecode is associated with a scalar field or an object reference field,

rewrite the bytecode to identify the bytecode as being associated with the scalar field when the bytecode is associated with the scalar field,

rewrite the bytecode to identify the bytecode as being associated with the object reference field when the bytecode is associated with the object reference field, wherein the bytecode rewriter is associated with producing a source file; a compiler arranged to accept the source file as input and produce an object file; and a virtual machine arranged to execute the object file.

7. In a computer system having a preloader coupled to a compiler and a virtual machine, a method for rewriting bytecodes to minimize runtime checks, comprising:

by the preloader,

determining whether a bytecode makes an active reference to a class which requires an execution of a static initializer;

determining if the class has a superclass which requires the execution of the static initializer, wherein the preloader produces a source file,

rewriting the bytecode to a new bytecode, by the preloader, which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer;

accepting the source file as input and produce an object file by the compiler; and executing the object file by the virtual machine.

8. (Canceled).

9. A method according to claim 7, further comprising:

rewriting the bytecode to a new bytecode, by the preloader, which explicitly indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer.

10. A method according to claim 7 further comprising:

rewriting the bytecode to a new bytecode, by the preloader, which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which has the superclass which requires the execution of the static initializer.

11. A method according to claim 7, further comprising:

rewriting the bytecode to a new bytecode, by the preloader, which explicitly indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which has the superclass which requires the execution of the static initializer.

12. A computer program product for rewriting bytecodes to minimize runtime checks in a computer system having a preloader coupled to a compiler and a virtual machine, comprising:

computer code for determining whether a bytecode makes an active reference to a class which requires an execution of a static initializer;

computer code for determining if the class has a superclass which requires the execution of the static initializer, wherein the preloader produces a source file;

computer code for rewriting the bytecode to a new bytecode, by the preloader, which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer;

computer code for accepting the source file as input and produce an object file by the compiler;

computer code for executing the object file by the virtual machine; and

a computer readable medium for storing the computer program product.

13. (Canceled).

14. A computer program product according to claim 12, further comprising:

computer code for rewriting the bytecode to a new bytecode, by the preloader, which explicitly indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which requires the execution of the static initializer.

15. A computer program product according to claim 12 further comprising:

computer code for rewriting the bytecode to a new bytecode, by the preloader, which indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which has the superclass which requires the execution of the static initializer.

16. A computer program product according to claim 12, further comprising:

computer code for rewriting the bytecode to a new bytecode, by the preloader, which explicitly indicates that at least one of the class and the superclass requires execution of the static initializer when it is determined that the bytecode makes the active reference to the class which has the superclass which requires the execution of the static initializer.